A REPORT ON CONTROL OF ACCESS TO STORED INFORMATION IN A COMPUTER UTILITY

A SPECIAL PROBLEM

Presented to the Graduate Faculty of the Department of Computer Science in North Texas State University in Partial Fulfillment of the Requirements

For the Degree of

MASTER OF SCIENCE

By

Mostafa Shakiba-Jahromi

Denton, Texas

1978

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

INTRODUCTION

Time-sharing computer systems permit large numbers of users to operate on common sets of data and programs. Since certain parts of these computer resources may be sensitive or proprietory, there exists the risks that information belonging to one user, may, contrary to his intent, become available to other users, and there is the additional risk that outside agencies may infiltrate the system and obtain information.

The question naturally arises of protecting one user's stored program and data against unauthorized access by others. Considerable work has already been done in providing protection against accidental access due to hardware malfunctions, but only recently the attention has been given to the protection of computations and information against those who attempt to gain access to private information without having proper authority. A central concern in protection is the authorization problem (Friedman, 1970) which involves determining whether a user who is recognized by the system should be allowed access to information he desires. The true authorization problem arises when several users are provided access to a common

data file; given the restriction that users are not permitted access to all data. Such a requirement may arise in police networks, government data centers, banking data centers, management information systems, or credit bureaus.

In this report, a solution to some of the problems concerned with the access control in a computer is defined and discussed. Special attention is concentrated on the problems of protecting both user and system information when shared procedure and data are permitted.

A set of processor access control mechanisms which were devised for the MULTICS system are described here. (A processor is a hardware device that is capable of executing a sequence of instructions.) MULTICS is a general purpose, multiple user, interactive computer system developed at project MAC of MIT (Corbato and Vyssootsky, 1965). In the course of MULTICS development, MAC researchers discovered that the processor provided only a limited set of access control mechanisms. The MULTICS system uses the concept of concentric protection rings to provide a variety of access rights (or level of authorization). The technique relies on the use of hardware and software facilities. In the MULTICS system for example, if a user has a file which in part contains sensitive data, he cannot merge all his data with that of his colleagues. He often must separate the sensitive data and save that in a

separate file; the common pool of data does not contain this sensitive and highly valuable data.

Although a great deal of effort has already been expended to protect computer resources from their environment, it is still true that no computer system has withstood determined efforts to bypass its internal protection controls by someone who has been given user programming access to the system. Such penetration efforts have been successful against virtually all available systems. Because of diversity and complexity of the problem, an adequate and comprehensive protection mechanism is not easy to design. This report does not intend to describe specific solutions to various protection problems; rather, it describes some mechanisms that support adequate solutions to protection problems and access control.

At the beginning, the general needs to control access to stored information in a computer utility are discussed here, and several criteria for comparing different sets of access control mechanisms are presented. (The computer utility is a generalization of the time-sharing concept which enables a central facility to dispense computer resources to users in an effective and efficient manner.) Considerable attention is given to the concept of protection mechanisms and the way these mechanisms are implemented to control access in a computer utility. Particular attention is focused on the concept of protection rings (Graham,

1968) as implemented in MULTICS, and the concepts of protection domains and capabilities relevant to access control.

A variety of other protection features such as passwords and cryptography have been included in most computer systems; however these other protection features have shown some obvious disadvantages and can be bypassed if the basic protection mechanisms are bypassed or broken. A variety of ideas and protection mechanisms are defined and discussed in this report. According to these discussions, the concepts of limited protection domains and capabilities are shown to be very effective and fundamental for enforcing protection of information and controlling access rights in a computer utility. Passwords and cryptographic techniques appear to exhibit some limitations and certain inefficiencies.

I. ACCESS MANAGEMENT IN A COMPUTER UTILITY

Protection of computations and information is an important aspect of a computer utility. The community of users of a computer utility certainly have diverse interests. It probably includes users who are competitive commercially. The system is used for many applications where sensitive data, such as company payroll records, need to be stored in the system. On the other hand, there are users in the community who wish to share with each

other; data and procedures. There are even groups of users working cooperatively on the same project. Service bureaus, software producing companies, and other service organizations have procedures which they wish to rent. A great potential benefit of a computer utility is its ability to allow users to easily communicate and cooperate. The role of protection in a computer utility is to control user interaction; guaranteeing total user separation when desired, allowing unrestricted user cooperation when necessary, and providing different users with flexible, but controlled, access to shared data and procedures. A good protection mechanism should not permit a user to interfere with other users. Without adequate protection, a dishonest user may alter the accounting procedure or data, thereby causing inequitable charges.

While there are many different modes of protection in a computer utility, most may be related to controlling access to stored information. Since stored information represents both data and procedures, control of access to stored information serves to regulate information processing as well.

General Properties of a Protection Mechanism

According to Graham (May, 1968), a satisfactory protection mechanism should have the following properties:

1. The capability to isolate one program from another, that is, a user should be able to

deny any access by other users to all of his segments. (A segment is a contiguous block of words containing a program, data, or both.)
2. The capability for a user to allow controlled access to his segments.
3. The capability to recognize different layers of protection in the same process. (A process is basically a program in execution.)
4. The capability for allowing procedures to be called across layers of protection without any special programming on the part of the calling procedure.

According to Schroeder and Saltzer (March, 1972) four criteria can be applied to a set of access control mechanisms to judge its usefulness in a computer utility: functional capability, economy, simplicity, and programming generality. The first criterion means that a set of access control mechanisms should be able to satisfy user protection needs in a natural manner (a quality which allows user interface. An obvious goal is to maximize this capability). The second criterion, economy, refers to the fact that the cost of specifying and utilizing a particular protection mechanism should be low enough that it is not an important consideration in determining the required level of access control. In addition, cost should be proportional to the functional capability used. The third criterion, simplicity, refers to the practice of designing the protection mechanism so simple that they may be completely understood. The fourth criterion, programming generality, means that the protection mechanisms should not control the way in which programs are organized and developed. It permits sharing

in such a way that encourages users to build upon one another's work. Obviously, it is not easy to design access control mechanisms which satisfy all of the above four criteria at the same time. Improvements in functional capability come at the expense of economy, simplicity, and programming generality. An important aspect in designing a set of access control mechanisms should be to maximize functional capability within the frame of the other three criteria. Protection mechanisms vary from system to system. In fact, they are sometimes different for different parts of the same system. For example, there are systems which have one protection mechanism for the file system, another for main memory, etc.

In computer systems, all information resides on some storage device. Hence, the information can be protected by dividing the address space (the collection of programs and data that are accessed in a process) of these storage devices into regions and allowing information to flow between regions in a controlled way. Each region of addressable space must be protected from unauthorized attempts by those who wish to obtain or alter information.

As an example, we consider protection in a file system. Each file should be protected from users who want to read the file or write into the file without presenting adequate authorization. Passing from one region of addressable space to another is similar to crossing a wall or a fence separating the regions. The walls are not penetrable, so, the

only way to cross a region is through the entries called "gates." Communication between the regions must be protected. That is, there must be a control mechanism at the gate to allow only authorized users to access the region. The walls which separate the regions of addressable space must be related to the addressing mechanism used in that particular system.

Passwords

In most existing file systems which are concerned with protection of information, passwords are used to enforce access control and to provide software protection for sensitive data (Madnick/Donovan, 1974). Associated with each file in the file directory is a password. Password schemes generally permit specific types of access to files. The user requesting access to a file must provide the correct password. If a user wishes to allow another user to access one of his files, he simply tells him the password for that file. In this environment, when a program attempts to open the file for input or output, a request is made to the operator to enter the correct password. If the operator cannot enter the correct password, then access to that file is denied. Therefore, the sensitive file can be protected. The use of passwords or prearranged set of questions requires no special hardware and is the least expensive means of user identification. This method has another

advantage in that only a small, fixed amount of space is needed for protection of information for each file. Its disadvantage is that passwords could be obtained by wire-tapping the communication links and other means. A dishonest systems programmer may be able to get all the passwords, since the protection information is stored in the system. Another disadvantage of passwords is that access control cannot easily be changed. If a user wishes to deny access of a file to a user who has the password, how does he do it? He can change the password, but now he has to inform all other users who are to be allowed access. Even if this were not the case, there are other reasons why password schemes as implemented to date do not properly solve the problem of access control in a large computer data base shared by many users.

One of these reasons is that passwords have been associated with files. In most current systems, information is protected at the file level only. Normally it is assumed that all data within a file have the same level of sensitivity. In real world situations this assumption is not necessarily true. Information from various sources is constantly coming into common data pools, where it can be used by all persons with access to that pool. Problems arise when certain information in a file should be available to some but not all authorized users of the file.

Cryptography

An alternative method for enforcing access control is the use of cryptographic techniques to provide protection for data transmission and to protect sensitive data files (Katzan, Computer Data Security, 1973). A common threat involved in tranmission of data is wiretapping. An effective method to prevent this is to encipher messages before transmission and decipher them after transmission. This technique prevents the disclosure of sensitive information. Sensitive files can be stored in an enciphered form to provide maximum protection against both accidental and malicious infiltration. All users may access the enciphered files but only authorized users know the way to deciper the contents. It should be noted that even the simplest method of encipherment can prevent accidental disclosure; at the same time, however, a more sophisticated cryptographic technique may be beaten by a professional intruder. There are many different approaches for enciphering/deciphering of messages. It can be performed by computer programs or by hardware devices depending upon the needs of particular computer installation and its users.

A cipher system consists of the following:

1. A set of rules that comprise the basic cryptographic process. This is called the general system.

2. A key (encoding sequences of characters), which may be variable and is chosen by the user for encoding the message.

The general system may be broken down into two classes:

1. Transposition methods; and
2. Substitution methods.

A transposition cipher involves a rearrangement of the characters of the original message in a prearranged manner. There are several transposition cipher methods. A simple and an elementary transposition cipher is one in which the characters of the original message are written in reverse order and then transmitted in groups of five characters. The characters lose their position but retain their identity.

Substitution cipher involves replacement of the characters of the original message by other characters. The characters retain their position but lose their identity. One of the advantages of using a computer program as an enciphering/deciphering mechanism is that the same program can be used to transform data files prior to being written on a storage medium, and for reversing the process when they are subsequently read.

It is not intended to discuss various ciphering techniques here, rather, the intention is to investigate the

effectiveness of cryptographic techniques for enforcing access control.

Cryptographic techniques have the advantage over the passwords method in that the code key does not have to be stored in the system, but the technique does impose the cost of encoding the files. The user needs to enter the code key only while he is enciphering or deciphering the file. Thus, there is no table that a dishonest system programmer can read to find all code keys or allow himself access. To discourage trial-and-error code breakers, the cryptographer should change keys at frequent intervals.

It should be noted that present-day communication cryptographic equipment is very expensive, and as a result it is not economically feasible to provide all the cryptographically-secure channels which are considered necessary. For example, it has been said that the cost of providing cryptographic security on every communication link carrying sensitive military traffic could exceed the total expenditure for the entire remainder of the system.

In the following sections, a set of hardware access control mechanisms that was devised in the course of MULTICS development is described. These mechanisms appear to provide a significant improvement in the simultaneous satisfaction of the four criteria mentioned earlier.

II. SEGMENTED VIRTUAL MEMORY ENVIRONMENT

In a virtual memory environment a job's address space no longer constrained by the size of physical memory, because the operating system produces the illusion of an extremely large memory by allowing information to reside on auxiliary storage devices, such as discs, tapes, drums, etc. Since this large memory is merely an illusion, it is called virtual memory (Donovan and Madnick, 1974).

In a system where users may share data in memory, it is essential to have control over access. There is a need to have a variety of access rights for each separate block of information. Current mechanisms allow memory to be subdivided into a large number of separate blocks of information called segments (Graham, May 1968). Each segment is a contiguous block of words containing a program, data, or both. It has a number of access control switches which specify various access privileges such as write/no-write, execute/no-execute, slave/master, etc. (Computers designed for multiprogramming usually have two modes of operations called master/slave or privileged/non-privileged.) This hardware features allows varying degrees of access to each segment.

An executing program in a segmented environment does not reference memory by abaolute address. Rather, the memory is composed of independent segments identified by numbers. All address space references require two components: (1) a segment specifier and (2) the location within the

segment. In a segmented environment, each job's address space actually consists of a collection of segments. The access rights are controlled through segment descriptors as shown in the following section.

A Protection Model

In this section, a model of hardware features which permits a satisfactory solution to the protection problems is described. A major component of this model is a segment (Katzan, 1973). In a computer with segment-addressing each word is addressed by an ordered pair of integers (S,W). S is the segment number and W is the word within the segment. Segment numbers range from 0 to a maximum allowable number of segments in a process; and the word number ranges from 0 to the current length of the segment to which it refers. Associated with each segment is a segment descriptor which contains location of the beginning of the segment, the current size of the segment, and the access control indicator (see figure 1).

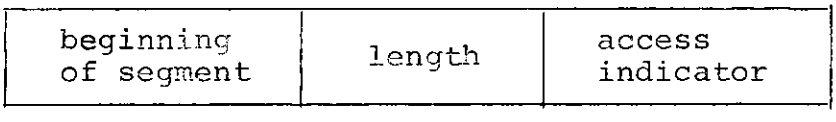

Fig. 1--Descriptor

The access indicator specifies whether the segment can be processed in slave mode, may be written, or may be executed. The access indicator also includes a "fault bit"

(Katzan, 1973) that causes an interrupt, whenever an unauthorized attempt is made to reference the segment. An interrupt is a hardware facility that causes the cpu to suspend execution, save its status, and transfers to a specific location which specifies the address of a program that is intended to take action in response to the interrupt (Mudnick, 1974). The following cases demonstrate the options that are permitted in processing a segment:

1. The segment is privileged procedure (when slave indicator is off and execute indicator is on).
2. The segment is writable data (when write indicator is on and execute indicator is off).
3. The segment is pure procedure (when write indicator is off and execute indicator is on). A pure or reenterant procedure is one that does not alter itself or contain data or any locations that are altered.

If the slave indicator is on, any procedure may access the segment, otherwise, only a master made segment (one with the master indicator on in its descriptor) may access it. If a fault bit is not zero, no access at all is permitted (interruption).

For every segment which a process may access, the corresponding descriptor resides in a description segment which is composed of segment descriptor words (SDWs). Each SDW is used to describe a single segment of the virtual memory, and the segment number of a segment used in an address is actually the index of its SDW in the descriptor segment. Thus, each user is assigned a collection of segments, called its virtual memory, and a special segment which

describes the segments that comprise the virtual memory. Among other things, a SDW contains the absolute address of the beginning of the corresponding segment in memory.

In any system there is a large number of descriptor segments. One descriptor segment exists for each active process (i.e., user or job), and whenever that process is executing, a hardware register reffered to as the "descriptor base register" contains the absolute address of the beginning of the corresponding descriptor segment in memory. The contents of dsecriptor base register indirectly defines that set of segments to which an executing process has access. Each processor contains logic for automatically translating two-part addresses into the corresponding absolute addresses. Changing the absolute address in the descriptor base register of a processor will cause the address translation logic to interpret two-part addresses relative to a different descriptor segment. This facility can be used to provide each user of the system with separate virtual memory. A single segment may be part of several virtual memories at the same time, allowing straightforward sharing of segments among users. In order to implement layered protection, the location counter and each descriptor is augmented with a field which contains a ring number as shown in figure 2.

beginning of segment	length	access indicator	ring number

segment descriptor word
in a segmented virtual memory

RING NUMBER	SEGMENT NUMBER OF EXECUTING PROCEDURE	CURRENT WORD NUMBER

Fig. 2--Location counter in a computer system using a segmented virtual memory and protection rings.

III. PROTECTION RINGS

Associated with each process are a limited number of domains called protection rings (a domain is a set of access rights which provides the means to protect procedure and data segments from other procedures that are part of the same computation). A ring (or more precisely, a protection ring) is a set of segments. The ring structure can be illustrated as concentric circles. In this way, segments of the memory containing sensitive information can be placed in a privileged ring together with programs that process, update, and extract this information. A subscriber enters a ring only at a carefully defined point, and once he enters the ring his activity is completely controlled. Various parts of the file have different conditions of access and different levels of privilege. The user must specify all of these, in addition to his name, problem number, account number, etc.

The MULTICS system, on which this ring concept has been implemented, is a general-purpose, multiple-user interactive

computer system developed at M.I.T.'s project MAC. These rings are numbered from 0 to a maximum allowable number in that particular system. The sets of access rights represented by the various rings of a process form, a collection of nested subsets, with ring 0 the largest set and the maximum numbered ring the smallest set in the collection. Thus, a process has the greatest access privilege when executing in ring 0, and it has the least access privilege when executing in the maximum numbered ring (Sultzer and Schroeder, March 1972).

Assigning a number to each ring is the mechanism by which access is controlled. Each segment is assigned to one and only one ring, when a procedure is executing in a segment, it assumes the ring number of that segment. The ring number determines its access level. The lower the ring number a procedure is executing in, the greater its access privileges. Rings numbered zero have the greatest privilege and are usually assigned to the supervisor. The ring structure is shown in figure 3 (page 19).

A procedure may access a segment with a segment number greater than its own ring number. Thus, if a procedure executes in ring m, then it has access to any segment n, where $m<n$, but it has no access to any segment K, where $K<m$, except with support of the supervisor program that executes in ring 0. In addition to the ring numbers, access control is managed through the access indicators mentioned above.

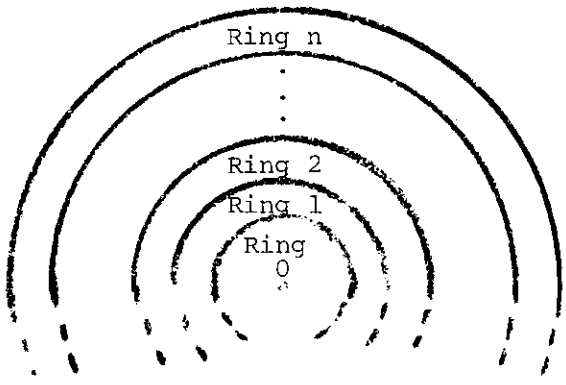

Fig. 3--Protection Ring Structure

Practical Considerations

Several practical considerations arise. The system should be aware of the passage of control from one ring to another. For example, when a procedure executing in ring m desires to transfer Cpu control to another segment with ring n, where M≠n, then the system should be aware of this change in control. When this happens, an interruption or a fault occurs. This fault is directed to the supervisor so that it may carry out appropriate housekeeping. This raises another consideration. When shared subroutines are used, all "calls" would have to pass through the supervisor or a copy of each routine would have to exist for each ring in its domain. Both of the above methods are inefficient. In order to solve this problem, these types of procedures are assigned to a consecutive set of rings, called the access bracket of the procedure. The access bracket is stored in the ring field of the segment descriptor word. The ring field of the descriptor will then

contain two integers specifying the lowest ring and the highest ring in the access bracket.

Now a transfer by a procedure in ring i to another procedure with access bracket (N1, N2) where $n1 \leq i \leq n2$ does not cause a fault and does not cause control to change rings, i.e., control remains in ring i. This means the value of the ring field of the location counter does not change. A reasonable and useful interpretation of the access bracket can be made for data segments. The following example clarifies the role of access bracket. Consider the segment D with access bracket (N1, N2). A procedure in ring i may:

1. Write into D if $i \leq n1$ (provided the write indicator is on).
2. Only read D if $n1 \leq i < n2$ (even if the write indicator is on).
3. Not access D at all if $i > n2$.

Supporting Software

In this section, a software necessary to support the satisfactory implementation of the protection mechanisms described before is briefly explained (Graham, 1968). We first consider the problem of allowing passage of control into an inner ring from an outer (higher numbered) ring. As it was mentioned earlier when a procedure executing in ring i attempts to transfer to another procedure with access bracket (N1, N2) and $i > n2$ a fault occurs. Since it is not usually desirable to allow transfer of control to a

procedure from an outer ring to an inner ring, the concept of access bracket is extended to include a third integer, n3, which defines the call bracket. The three integers are referred to as the ring bracket. When a procedure executing in ring i attempts to call a procedure, p with ring bracket (N1, N2, N3), the following cases are identified:

1. If $n2 < i \leq n3$ the call is permitted only to specific entry points in p. (The call bracket is implemented by software.)
2. If $i > n3$ the call is not permitted at all.
3. If $i > n2$ a fault occurs. The fault handler sorts out this case for $n2 < i \leq n3$.

A key component in the implementation of protection using ring structures is ring 0 routine called the gatekeeper that is used by the system to manage interring crossings and returns. It checks to see whether access should be allowed or denied.

The use of protection rings is a technique for achieving access control and protection in a multiple user environment. Protection rings offer the potential for increased versatility so that protection can be more closely matched to the needs of the users of the system.

Additional Considerations of Protection Rings

The association of multiple domains of protection with a process (a process may be executing in several protection rings) generates the need for a new kind of access capability; the capability to change the domain of execution of a process. This changing of domain provides

additional access capabilities available to a process and must be carefully controlled. In other words, switching the ring of execution to lower numbered ring makes additional access capabilities, while switching the ring of execution to a higher numbered ring reduces the available access capabilities. A domain provides the means to protect procedure and data segments from other procedures that are part of the same computation. By using domains, it should be possible to make certain access capabilities available to a process only when particular programs are being executed. Restricting the start of execution in a particular domain to certain program locations, called gates, provides this ability, because it gives the program sections that begin at those locations complete control over the use made of the access capabilities included in the domain. Thus, changing the domain of execution must be restricted to occur only as the result of a transfer of control to one of these gate locations of another domain. A process changes its protection status by changing the domain out of which it operates.

With a completely general implementation of domains, each domain could provide protection against the procedures executing in all other domains of a process. The corresponding property of rings is that the protection provided by a given ring of a process is effective against procedures executing in higher numbered rings. Switching the ring of

execution to a lower numbered ring makes additional access capabilities available to a process, while switching to a higher number reduces the available access capabilities. Thus, the downward ring switching capability must be coupled to a transfer of control to a gate into the lower numbered ring.

In MULTICS, there is a hierarchy of files and programs. Each segment (program or file) has clearance level. This level is placed in a hardware register when the program is in control. If the program calls another program (or accesses a file) which is not at the same security level or a lower level, a hardware interrupt is taken. The interrupt routine must then examine a gate list which contains the list of acceptable users who may access this segment. Thus, when an access request is made to a segment, the gatekeeper program checks to see that the requestor is on the segment's gate list. If not the security levels of the two segments are compared, and the requestor is permitted access only if his programs have a sufficient level of clearance. The gatekeeper distinguishes types of access as read, write, and execute. In this sytem, there is a security matrix which is stored in the form of gate lists, which indicate whether user i can access datum j for read/write/execute purposes.

The following section gives an extensive discussion of protection domains which provide a better understanding of the kinds of control required.

IV. PROTECTION MODELS AND PROTECTION DOMAINS

A protection model views the computer as a set of active elements called Subjects and a set of passive elements called Objects (Graham and Denning, 1972). The protection model defines the access rights of each subject to each object. This protection model can be represented in the form of a protection matrix as shown in figure 4.

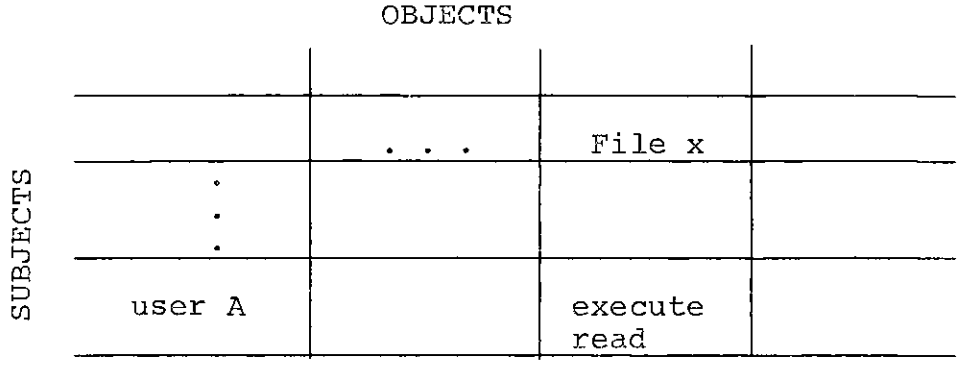

Fig. 4--A Protection Matrix

The rows of this matrix correspond to the users of the system (subjects). The columns of the matrix correspond to the objects. For each subject/object pair, the corresponding entry in the matrix defines the set of access rights that the subject has to object. Figure 4 shows that user A (Subject), may read or execute file x (Object). Access rights in this protection matrix also control changes to the protection matrix itself; for example, a user with "Delete" access to a file is able to eliminate the particular file from the protection matrix. A protection domain defines the set of access rights that one subject has to the objects

of the system. Protection domains are represented as rows of the protection matrix. The following PL/1 data structure represents an example of a typical data file and its corresponding security matrix which grant access right selectively to different users within the system (Conway and Maxwell, April 1972).

```
DECLARE 1 EMPLOYEE (1000)
          2 SOC_SEC_NO
          2 NAME
          2 ADDR
            3 STREET
            3 CITY
            3 STATE_ZIP
          2 SALARY
            3 PREV
            3 CURRENT
          2 PERFORMANCE
            3 CLASS
            3 INDEX
          2 MEDICAL
            3 PHYSICAL
            3 DOCTOR
              .
              .
              .
```

Fig. 5--A typical file represented as a PL/1 array of structures.

USERS	SOC_SEC_NO	NAME	ADDR	SALARY CURRENT	PERFORMANCE CLASS	PERFORMANCE INDEX	MEDICAL
A	R	R	R	R,W	R,W	R,W	R
B	R	R	R	R.W	N	N	N
C	R	R	R	R,W	R,W	R,W	R
D	R	R	R	N	N	N	R,W
E	R	N	N	N	N	N	N
F	N	N	N	N	N	N	N

Key: R--Read access
W--Write access
N--No access

Fig. 6--Security matrix for the file represented in Figure 5.

In most recent computer systems subjects represent basically the authorized users of the system. In these systems, the supervisor or operating system is another subject which has total access to all objects in the system. In these systems, every subunit of a user's program executes in the same protection domain, and that protection domain has access rights to all objects that a user ever needs. With this model of protection, it is not easy to limit the access rights of specific subprograms executed on behalf of a user. A small subunit of a program typically only needs access to a small number of objects. If small subunits of a program execute in their own protection domains, then the protection domain can be kept limited and relatively small. A large program usually needs access to many objects. Thus, protection domains can be kept small and limited only if a large program executes in many different protection

domains and constantly switches between these protection domains during its execution.

Protection Domain Switching

A change in the domain of execution of a process can occur only when the executing procedure transfers control to a gate of another domain (Saltzer and Schroeder, 1972). In the context of most programming languages, whenever a transfer occurs within the procedure, this represents a subroutine call, a return following a call, or a non-local goto. All of these three operations produce a change in the environment of the execution point. Each procedure could have its own protection domain, although every procedure call does not necessarily involve a domain switch. The phrase, "protected procedure," is used when it is necessary to emphasize that the procedure call does involve a domain switch. A protected procedure appears as both a subject and an object in a protection matrix. It is an object because other subjects may have the right to call it. The right to call a procedure requires a special access right such as an "enter" right to the procedure. The protected procedure is also a subject in the protection matrix because it executes in its own protection domain. The call operation has the additional function of transmitting arguments and recording a return point. Whenever a call takes place, the execution begins in the protection domain

of the called procedure. A return instruction triggers a return to the previous protection domain. Performing these functions generally requires the cooperation of both the procedure initiating the operation and the procedure receiving control. The domain switch is simplified if there are no access rights passed as parameters in the call. Figure 7 illustrates a simple domain switch. In this situation, user A, executing in his basic domain, can call Editor. A dictionary (the dictionary may contain descriptions of the files, report, and other entities known to the system, as well as the security information) can only be read while executing in the editors domain. (The security matrix is contained in a special dictionary, which is created by the user.) The user can read or write files x and y either from his basic domain or after calling the editor; however, he can use the dictionary to check the files for spelling mistakes only when he has transferred control to editor.

OBJECTS

SUBJECTS	Editor	File x	File y	Dictionary
⋮				
User A	Enter	Read Write	Read Write	
Editor		Read Write	Read Write	Read

Fig. 7--Simple Domain Switch

The situation becomes more complex if access rights to objects are to be passed as parameters and if the protected procedure is to be reenterant. (A procedure is reenterant if it has completely separate data and procedure sections. Reenterant code can be stored among several users by giving each user a private copy of the data section and allowing all users to share a single copy of the procedure section.) In this case, the call of the protected procedure results in the creation of a new protection domain; conceptually meaning that a new row is created in the protection matrix. This new protection domain contains both the permanent access rights of the protected procedure (these are defined by a template domain associated with the procedure) and the access rights that are passed as parameters in the call. The new protection domain is destroyed by the return from the protected procedure. Figures 8 and 9 illustrate this situation.

OBJECTS

SUBJECTS	Editor	File x	File y	Dictionary
⋮				
User A	Enter	Read Write	Read Write	
Editor Template				Read
⋮				

Fig. 8--Protection matrix before call to editor.

OBJECTS

SUBJECTS	Editor	File x	File y	Dictionary
⋮				
User A	Enter	Read Write	Read Write	
Editor Template				Read
Instance of Editor		Read Write		Read

Fig. 9--Protection matrix during call to editor.

In figure 8, the user is executing in his basic domain, and the editor's template domain only has the right to read the dictionary. If the user then calls the editor in order to edit file, x, he passess access rights for file x to the editor. This creates a new domain labeled "instance of editor," in figure 9. If file y is sensitive, the user does not have to allow the editor access to it, and the editor can protect the dictionary from direct access by the user. The domain change for a reenterant protected procedure seems to be cumbersome when it is explained in terms of an abstract protection model. Later on, an implementation is suggested that allows protection domain to be created and destroyed easily and efficiently. The following sections discuss and illustrate the concept of capabilities and their significance in the design of protection mechanisms relevant to access control.

V. CAPABILITY-BASED ADDRESSING

Capability may be thought of as a uniform method of addressing and protecting access to shared segments and procedures. A capability may also be considered as a ticket whose possession confers access privileges for the stored object (segment). Capability-based addressing (Fabray, 1974) is a way to address and control access to objects even when the objects are stored in the primary memory of a computer system. Different systems use capabilities in quite different ways. Capabilities generally have the following properties:

1. Capabilities are system-protected names for objects (segments). A subject has access to an object only if it possesses a capability for that object. Capabilities are permanent and can be stored in programs or in the auxiliary memory for an indefinite period.
2. A part of the capability determines the access rights that the capability allows to the object that it names.
3. When an object is created, a capability for that object is created. Any subject in possession of a capability has some freedom to move it, to copy it, or to pass it as a parameter to procedures in different domains, but it is not allowed to modify it.

The creator of the object may give a copy of the capability to other subjects. Recipients of a copy of a capability may use it to access the object, or they may make other copies of it to give to other subjects. When a capability is given to another subject, the access rights of the capability may be restricted. Therefore, each copy of a capability may allow varying access rights to the object.

Capabilities are similar to descriptors as implemented in MULTICS. Several systems have used capabilities to facilitate sharing and protection of objects that are not loaded in primary memory. In these systems, interpretation of capabilities is done by software, and the primary memory is addressed and controlled by whatever means is available. Calls to the system software are needed in order to use a capability or switch to a different protection domain.

The following explanation of capability-based addressing assumes that memory is organized into segments, where a segment is a variable length sequence of memory words. A word in a segment is addressed by supplying an identifier (a number) for the segment and an offset that specifies the particular word of the segment. A descriptor as implemented in MULTICS and the Burroughs Systems, is a protected identifier that points to a segment. The descriptor also specifies the access rights that are allowed to the segment. An instruction references a memory word by pointing to a descriptor for the segment and by providing an offset to specify the desired words of the segment. Access rights of the segment descriptor are used to deny undesired access to segment. Capabilities are almost identical to descriptors. They perform the same function of identifying the segment and specifying the access rights to the segment. The primary difference between capabilities and descriptors arises because descriptor-based systems usually provide little

freedom to manipulate the descriptors (only hardware and low level of system software control all movements of the descriptors), while capability-based systems allow the capabilities to be moved and copied and passed as parameters. This freedom to manipulate capabilities greatly simplifies the implementation of parameter passing during a domain switch.

Implementations for Capability-Based Addressing

A capability usually consists of an identifier that can be used to find the object, a field which defines the type of the object, and a field defining the access rights. Figure 10 illustrates a capability that allows only read access to a segment. The access rights field is probably a set of bits--one bit for each mode of access. The interpretation of these bits depends on the types of the object.

IDENTIFIER	Type of the Object	Access Rights*
Pointer to the Segment	Segment	Read

*A set of bits--one bit for each mode of access.

Fig. 10--Internal Structure of a Capability

Control over capabilities is necessary to prevent a user from creating a capability which can be used later on to give unauthorized access to an object. In order to

achieve this type of control, two approaches exist:

1. Capabilities should be stored always in special locations such as capability segments and capability registers.
2. An extra tag bit should be included with each memory word. The tag bit must be inaccessible to the user. It identifies whether the word contains a capability, and the hardware then controls the modification of the words that are identified as capabilities.

The identifier in a capability can be implemented in two different ways:

1. The identifier may be a pointer to the object--it may contain the address for the object.
2. The identifer may be a unique code that is permanently associated with the object. This is called a unique identifer.

The pointer approach makes it simpler to use the capability to reach the object. Each time the identifier points directly to the object, then it must be updated whenever the object is moved. If the capabilities are not updated properly, then a capability for one object may end up pointing to a different object.

The second approach, based on unique identifier, makes it unnecessary to keep track of capabilities and to update them. A unique identifier cannot be reused unless all capabilities for the previous object have been destroyed. The unique identifier approach requires that the current address of the object must be determined from the unique identifier; each time the capability is used to address the object. This would be implemented by maintaining a

large hash table to associate the current address of objects with the unique identifier of the capabilities.

Implementing Limited Protection Domains

A capability corresponds to a set of access rights for a single object in the protection model. A protection domain, which is a row of the protection matrix, is realized as the set of capabilities that are accessible to the subject.

There are two most frequently used types of capabilities which are often given special treatment. A storage capability grants access to a segment in the virtual memory of a process. Its access code may specify read, write, or execute permissions. An enter capability grants permission to invoke a procedure in a domain differing from that of its caller.

Figure 11 illustrates a part of a protection matrix and the corresponding capabilities. In this figure, user A can call the editor and pass access rights for file x by passing a copy of the capability for file x. The next section describes the implementation of efficient switching between protection domains.

OBJECTS

Subjects/Objects	Editor	File x	File y	Dictionary
⋮				
User A	Enter	Read Write	Read Write	
Editor Template				Read

Capabilities of User A

ID for Editor	PROC	Enter
ID for File x	File	Read/Write
ID for File y	File	Read/Write
Capabilities of Editor Template		
ID for Dictionary	File	Read

Fig. 11--Protection matrix stored as capability.

VI. CAPABILITY-BASED IMPLEMENTATION OF EFFICIENT DOMAIN SWITCHING

With capability-based addressing, it is reasonably straightforward to implement domain switching as part of the hardware implementation of the call and return operations. With appropriate hardware support, the overhead to switch projection domains could be comparable to that of a simple procedure call in existing computer systems. The capability automatically provides access authorization to

the object and enforces limitations on the authorized access.

The most efficient implementation for domain switching is probably achieved by using stacks (Neuman, 1975). The process stack is divided into frames. At any time in its execution, the process has access only to the stack frame associated with the most recent protected procedure activation. In calling another protected procedure, parameters for the call are pushed into the stack, and the call instruction delimits the new stack frame to be used by the calling procedure. This situation is illustrated in figure 12. In this figure, a call occurs to an editor, and the frame markers are used to delimit the stack frames. After the call to the editor, only that part of the stack above the highest stack frame marker is accessible. Parameters may be data or capability.

When the editor issues a return instruction, the editor's stack frame is deleted--except for any return data or capabilities. The return data is left on the top of the stack. Figure 13 shows the state of the stack after return from the editor. If the editor has copied a capability for the dictionary into the stack, then this copy of that capability is automatically deleted by the return instruction.

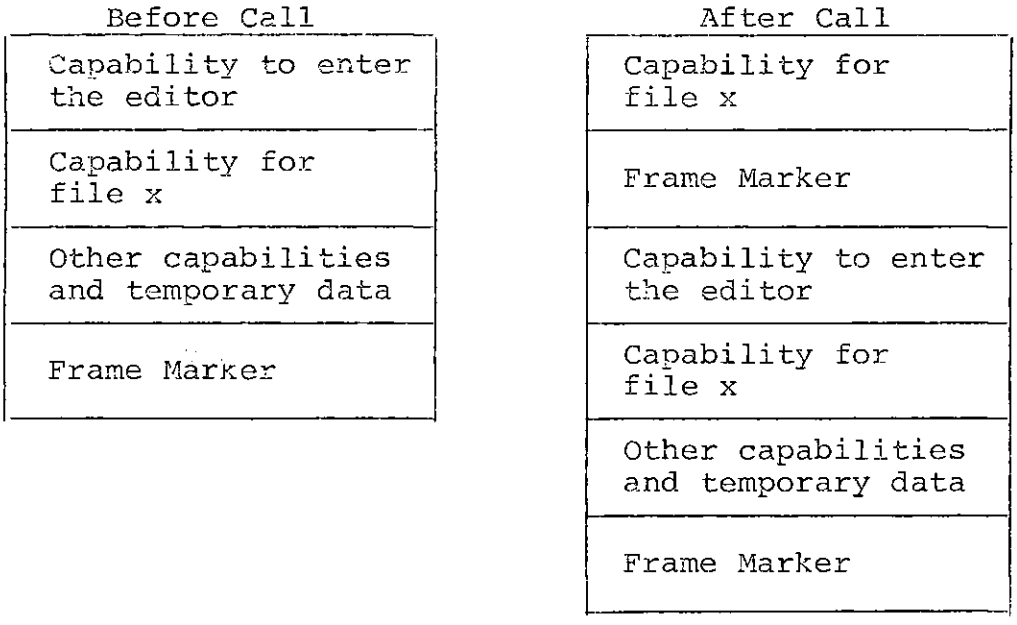

Fig. 12--State of the stack before and after a call to the editor.

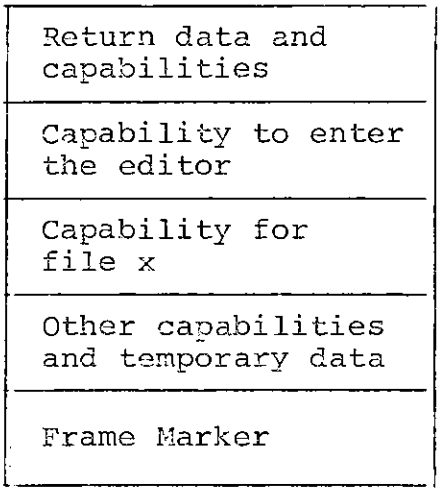

Fig. 13--State of the stack after return from the editor.

VII. DIRECTORIES FOR THE STORAGE AND SHARING OF CAPABILITIES

In a protection system which allows a large number of independent protection domains, the protection domain

must be stored and maintained efficiently. If each protection subject had to store large lists of capabilities, one for each object, it is allowed to access, then the maintenance of all this information could be a serious problem.

There must be provisions for controlled sharing of capabilities between users of the system. If capabilities are stored in data segments, then any segment can be used to store and share capabilities. To maintain control over capabilities, most long term storage and sharing should be handled by a system of directories that are designed for these purposes.

A directory is basically a table of enteries that associate user chosen names with capabilities. Directories can have three distinct roles in a capability-based system:

1. They allow objects to be addressed by user-chosen names rather than by the system-generated capabilities.
2. They guarantee the existence of at least one capability for every existing object.
3. They simplify the storage and maintenance of the information required to implement a protection matrix and preserve the capability of inactive users.

A subject with access to a given directory is allowed to request the capability associated with a given name. To facilitate controlled sharing, it is desirable to have a means of allowing subjects access to some of the capabilities stored in a directory without necessarily allowing them access to all the capabilities in the directory. MULTICS accomplished this task by using access control

lists (Saltzer, July 1974) which provide a list of all users who may access the segment. Tne enteries in the list give the access attributes for every domain which has access privileges to the particular object. For each object, the subject which created that object provides a procedure and an access control list. A generalization of this approach has been suggested based on the idea of locks and keys. In every domain, there is a list of objects together with a bit pattern for each object; the bit pattern (the key) serves as an identification for the access privileges to the object. Associated with the object there is a list of unique bit patterns (locks) together with associated access privileges for each bit pattern. A request to the directory system requires both a capability for that directory and a key. The request is fulfilled only if the key matches a lock that has been associated with the named entry in the directory. The key can be implemented as a capability. In this case, the capability is simply an identifier and its modification is not allowed except to limit its access rights. Directories, themselves, are protected objects of the system, and a specific directory can be accessed only by a subject possessing a capability for that directory. For security it would be useful if the system could guarantee that the directory system is the only means to share capabilities between distinct users or to store them for relatively long periods of time. This makes it much easier to control and to

monitor the security status of the system. Directories are also useful as a way of modifying protection domains when users share access to objects. Thus, directories play a key role in storing and maintaining protection domains. (Each subject's protection domain includes all the capabilities that the subject can retrieve from the directory system).

Correct Implementation of Protection

Correct implementation of the basic protection mechanism is critical to all security. If the basic protection and addressing mechanisms are broken or bypassed, other elements in the system will be in danger. Thus, the correctness of the protection mechanisms must be guaranteed with a very high degree of confidence. The implementation of a very flexible protection system is more complex and more difficult than the implementation of a more rigid and limited protection system. The concept of limited protection domains as implemented in MULTICS which was mentioned earlier, makes it easier to control the interactions between different system modules. Furthermore, capability-based addressing simplifies some of the system software. In order to encourage good programming practices and to support the concept of limited protection domains, protection mechanisms should not prevent a program from referring subtasks to other protected procedures. Thus, a program should

be able to pass any of its own access rights to another protected procedure. On the other hand, there are situations where one user of the system must be prevented from making access rights available to another user.

Suggestions

During the process of devising an access control method, the following principles are suggested to be kept in mind:

1. The method should allow efficient control of individual data elements (rather than of files or records only).
2. The method should not extract unwarranted cost in storage or elsewhere from the user who wants only a small portion of his data protected.
3. The method should be independent of both machine and file structure.
4. Finally, it should be sufficiently modular to permit cost-effectiveness experiments to be undertaken.

CONCLUDING REMARKS

This report focuses on ideas and concepts that support protection of information in a computer utility. Several mechanisms and techniques which control access to stored information are discussed and illustrated. The idea of using password schemes and cryptographic techniques as methods of enforcing access control and providing software

protection for sensitive data, is not a new one. However, these methods although useful and economically feasible for certain applications, do not satisfy the needs and requirements of today's time-sharing concept, where large numbers of users operate on common sets of data and programs.

There are several reasons why present password schemes do not properly solve the problem of access control in a time-sharing environment. One of these reasons is that passwords usually have been associated with files. In most current systems, information is protected at the file level only. It is normally assumed that all data within a file are at the same level of sensitivity. In real world situations, this assumption is not necessarily true. Information from various sources which comes into common data pools, can be used by all of the users who have access to that pool. Problems arise when certain information in a file should be only available to some but not all authorized users of the file. Another disadvantage of the password method is that a dishonest systems programmer may be able to get all the passwords, since the protection information is stored in the system. There are still other problems with password methods which make them inefficient and disadvantageous.

The use of protection rings is another technique for achieving access control and protection in a multiple user environment. The protection provided by a given ring of a process is effective against procedures executing in

higher numbered rings. Switching the ring of execution to a lower number ring makes additional access capabilities available to a process, while switching to a higher number reduces the available access capabilities. Thus, the downward ring switching of capability must be coupled to a transfer of control to a gate into a lower numbered ring. The concept of call bracket as implemented in MULTICS, efficiently manages the transfer of control between protection rings and checks to see whether access should be allowed or denied.

The concept of limited protection domains as explained before, is another effective technique for enforcing access control. This technique limits and restricts the access rights of specific subprograms executed on behalf of a user. This restriction is necessary, otherwise every subunit of a user's program may execute in the same protection domain, and that protection domain has access rights to all subjects that a user ever needs. Furthermore, a small subunit of a program typically only needs access to a small number of objects. If small subunits of a program execute in their own protection domains, then the protection domain can be kept limited and relatively small. Limited protection domains allow each subunit or module of a program to be executed in a restricted environment that can prevent unanticipated or undesirable actions by that module.

A large program usually needs access to many objects. Thus, protection domains can be kept small and limited only if a large program executes in many different protection domains and constantly switches between these protection domains during its execution. With a completely general implementation of domains, each domain could provide protection against the procedures executing in all other domains of a process.

Another approach to enforce access control and to provide software protection, is based on the concept of capabilities incorporated into the addressing structure of the computer. Capability-based addressing is a way to address and control access to segments even when the segments are stored in the primary memory of a computer system. This technique is seen as a particular way to support future requirements for protection without sacrificing for flexibility and sharing.

A comparison of internal structure of capabilities and descriptors (figures 1 and 10), shows that the two techniques are basically similar. They both perform the same function of identifying the segment and specifying the access rights to that segment. Descriptor-based systems, as implemented in MULTICS, make use of access control lists to associate permissions with objects and to check authorization on each access, while a capability-based system, makes use of capability lists to associate permissions with

domains and checks authorization only once, whenever capabilities are granted. The reduced need for checking contributes to run time efficiency in the capability-based method. To facilitate controlled sharing, it is desirable to allow subjects access to some but not all of the capabilities which are stored in the directory.

The primary difference between capabilities and descriptors arises because descriptor-based systems usually provide little freedom to manipulate the descriptors (only hardware and a low level of system software control all movements of descriptors), while capability-based systems allow the capabilities to be moved and copied and passed as parameters. This freedom to manipulate capabilities greatly simplifies the implementation of parameter passing during a domain switch, and furthermore it simplifies some of the system software.

In short, the capability-based addressing technique appears to be the simplest and the most thorough approach to enforce limited protection domain and to control access rights in a computer utility. Many ideas concerning protection in computer systems are still the subjects of basic research, and before they are put into practice, they need to be examined more. A breakthrough in protection will not be easy to achieve. Any single idea may not succeed if it is not properly supported by other equally new ideas. It is a major undertaking to achieve an effective

combination of those ideas. Although improved security and protection usually involves higher costs, the new ideas promise to promote security and lead to the development of a better software and reduce the costs at the same time.

REFERENCES

Corbato, F. J. and V. A. Vyssootsky, "Introduction and Overview of the Multics System," Proceedings, AFIPS, 1965 Fall Joint Computer Conference, Vol. 27, Part 1, New York, pp. 185-196.

Conway, R. W., W. L. Maxwell, and H. L. Morgan, "On the Implementation of Security Measures in Information Systems," Communications of the ACM, 15 No. 4 (April, 1972), 211-220.

Donovan, J. J., Systems Programming, New York, McGraw-Hill Book Company, 1972.

Fabray, R. S., "Capability-Based Addressing," Communications of the ACM, 17, no. 7 (July, 1974), 403-412.

Friedman, T. D., "The Authorization Problem in Shared Files," IBM Systems Journal, 9, No. 4 (1970), 258-280.

Graham, G. S. and P. J. Denning, "Protection-Principle and Practice," in Proceedings, AFIPS 1972 Spring Joint Computer Conference, Vol. 40, 1972, pp. 417-424.

Graham, R. M., "Protection in an Information Processing Utility," Communications of the ACM, 11, No. 5 (May, 1968), 365-269.

Harrison, A. and L. Ruzzo, "Protection in Operating Systems," ACM, 19, No. 8 (August, 1976), 461-466.

Hoffman, L. J., "Computer and Privacy," Computing Surveys, 1, No. 2 (June, 1969), 85-90.

Katzan, H., Computer Data Security, New York, Van Nostrand Reinhold Company, 1973.

_____, Operating Systems: A Pragmatic Approach, New York, Van Nostrand Reinhold Company, 1973.

Madnick, E. S. and J. J. Donovan, Operating Systems, New York, McGraw-Hill Company, 1974.

Neumann, P. G and L. Robinson, "A Probably Secure Operating System, Stanford Research Inst. Final Report, Menlo Park, California, June 1975.

Saltzer, J. H., "Protection and the Control of Information Sharing in Multics," Communications of the ACM, 17, No. 7 (July, 1974), 388-402.

_____ and M. D. Schroeder, "A Hardware Architecture for Implementing Protection Rings," Communications of the ACM, 15, No. 3 (March, 1972), 157-165.